# Data Flash Programming and Calibrating the bq20z70 and bq20z90 Family of Gas Gauges

*Doug Williams*                                                                                   *Battery Management*

### ABSTRACT

This application report presents a strategy for high-speed, economical calibration and data flash programming of the bq20z70/bq20z90 advanced gas gauge chipset family. VB6 code examples are provided, along with step-by-step instructions for preparing a golden battery pack.

**Contents**

## 1 Introduction

The bq20z70/bq20z90 family of advanced gas gauges is built with new technology and a new architecture for both data flash access and calibration. With this new architecture, unit production cost and capital equipment investment can be minimized, as there is no longer a need to perform a learning cycle on each pack. A single "golden pack" can become the source of data for all other packs. A method is shown to quickly read and write the golden image. Also, the calibration method is quick and simple because most of the calibration routines are built into the firmware of the target device.

The methods in this document are presented as VB6 (Visual Basic 6) functions. These functions were copied directly from working code. In order to read from and write to the data flash, they use five types of SMBus read and write functions. These can be duplicated in any software environment that has SMBus communication capabilities. As used herein, each Read/Write function is designed for communication with a gas gauge, so the device address (0x16) is omitted for clarity.

1. WriteSMBusInteger() has two arguments – the SMBus command and a signed integer. Internally, this function separates the integer into two bytes for transmission by the SMBus write-word protocol.
2. WriteSMBusByteArray() has three arguments – the SMBus command, the array of bytes and an integer specifying the length of the byte array. Internally, this function separates the byte array into separate bytes for transmission by the SMBus write-block protocol.
3. WriteSMBusCommand() has only one argument – the SMBus command.
4. ReadSMBusUnsignedInteger has two arguments – the SMBus command and the returned integer.
5. ReadSMBusByteArray() has three arguments – the SMBus command, the returned array of bytes, and the returned length of the byte array. It is internally implemented with the SMBus read-block protocol.

Also used in these functions is a simple delay routine called DoDelay. VB6 code for this procedure is provided at the end of the document.

Impedance Track is a trademark of Texas Instruments.

Error handling is not implemented in this sample code, because requirements are unique and varied. Also, constants are hard-coded into the functions to improve clarity rather than documenting them in code elsewhere as would normally be good coding practice.

A good strategy for production is a seven-step process flow:

1. Write the data flash image to each device. This image was read from a *golden* pack.
2. Calibrate the device.
3. Update any individual flash locations, such as serial number, lot code, and date.
4. Perform any desired protection tests.
5. Connect the cells.
6. Initiate the Impedance Track™ algorithm
7. Seal the pack.

In this document, the first three steps are examined in detail.

## 2    Preparing the Golden Pack

Impedance Track™ technology allows the bq20z70/bq20z90 gas gauge to automatically acquire and maintain parameters for battery modeling needed for continuous accuracy, regardless of battery model or manufacturer. The ICs are shipped preprogrammed with default values for these parameters. In the course of daily use (charge, discharge, unused), the algorithm collects new parameters. Parameter acquisition is complete after one full discharge cycle and subsequent relaxation takes place.

The default parameters that are used for fuel gauging prior to discharge activity are less accurate than parameters acquired during such activity. Therefore, the error of the gas gauge is more than the 1% that is achieved after parameter acquisition. It is desirable to have optimal accuracy in the battery packs coming from the production line even before any discharge activity occurs. This can be accomplished by performing a discharge cycle on one battery pack (let it acquire optimized parameters), save its data flash in a file, and then program the golden data into all battery packs coming from the production line.

### 2.1    Creating Pre-Learned Defaults

1. Assemble a battery pack with the bq20z80 solution, which includes setting basic flash constants for a given pack configuration, calibrating the pack, connecting *System Present* to ground, and enabling IT. This is described in detail in the application report *Pack Assembly and the bq20z80* (SLUA335), which also applies to the bq20z70 and the bq20z90.
2. In particular, it is important to set parameters specific to the number of serial cells used. This is described in application report *bq20z80 EVM Data Flash Settings for Number of Serial Cells and Pack Capacity* (SLVA208),which also applies to the bq20z70 and the bq20z90 .
3. To achieve maximum accuracy of first cycle parameter acquisition, set an initial guess for Qmax Cell 0, Qmax Cell 1, Qmax Cell 2, Qmax Cell 3 and Qmax Pack. These values are in mAh as specified in the battery manufacturer data sheet. For example, if single-cell data-sheet capacity is 2400 mAh and 3 parallel cells are used, set each value to $2400 \times 3 = 7200$ mAh.
4. Charge the pack to full.
5. Let it relax for 2 hours.
6. Discharge the pack to the minimum system-acceptable voltage (should be the same as DF.Gas Gauging.IT Cfg.Term Voltage) at the typical application rate. The exact rate is not critical.
7. Let it relax for 5 hours.
8. Repeat steps 4 through 7 to achieve maximum impedance table accuracy. Verify that DF.Gas Gauging.State.Update Status reads 06. If not, repeat the cycle. Its normal value should be 06.
9. Use the EVSW to export the .gg File. Open the .gg file with Notepad to change DF.Gas Gauging.State.UpdateStatus to 02. Change DF.SBS Configuration.Data.Cycle Count to 0.
10. Reprogram the pack with a fresh .senc to clear all hidden constants.
11. Use the EVSW to import the modified .gg file as saved in step 9. Write All.
12. Send reset command (0x0041).

The *golden pack* is now ready to have its data flash read into a binary file as described in the function listed in Section 3.

## 3    Reading and Saving the Data Flash Image From the Golden Pack

Note that this step only needs to be done once for a given project.

```
Function SaveDataFlashImageToFile(sFileName As String) As Long

    Dim iNumberOfRows As Integer
    Dim lError As Long
    Dim yRowData(32) As Byte
    Dim yDataFlashImage(&H700) As Byte
    Dim iRow As Integer
    Dim iIndex As Integer
    Dim iLen As Integer
    Dim iFileNumber As Integer

    '// FOR CLARITY, WITHOUT USING CONSTANTS
    '// 0x700 is the data flash size.
        0x700 \ 32 = 56 rows
    iNumberOfRows = &H700 \ 32

    '// PUT DEVICE INTO ROM MODE
    lError = WriteSMBusInteger(&H0, &HF00)
    DoDelay 0.01

    '// READ THE DATA FLASH, ROW BY ROW
    For iRow = 0 To iNumberOfRows - 1

        '// Set the address for the row. &H9 (0x09) is the ROM mode command.
        '// 0x200 is the row number where data flash starts.
        '// Multiplication by 32 gives us the actual physical address where each row starts
        lError = WriteSMBusInteger(&H9, (&H200 + iRow) * 32)

        '// Read the row. &HC (0x0c) is the ROM mode command.
        lError = ReadSMBusByteArray(&HC, yRowData, iLen)

        '//Copy this row into its place in a big byte array
        For iIndex = 0 To 32 - 1
            yDataFlashImage((iRow * 32) + iIndex) = yRowData(iIndex)
        Next iIndex

    Next iRow

    '// WRITE DATA FLASH IMAGE TO FILE
    iFileNumber = FreeFile
    Open sFileName For Binary Access Write As #iFileNumber
    Put #iFileNumber, , yDataFlashImage
    Close #iFileNumber

    '// EXECUTE GAS GAUGE PROGRAM
    lError = WriteSMBusCommand(&H8)

End Function
```

## 4    Writing the Data Flash Image to Each Target Device

The following method is fast. It only takes about 2 seconds to write the entire data flash in this manner.

> **CAUTION**
>
> If power is interrupted during this process, the device may become unusable.

```
Function WriteDataFlashImageFromFile(sFileName As String) As Long

    Dim lError As Long
    Dim iFileNumber As Integer
    Dim iNumberOfRows As Integer
    Dim iRow As Integer
    Dim iIndex As Integer
    Dim yRowData(32) As Byte
    Dim yDataFlashImage(&H700) As Byte

    '// READ THE FLASH IMAGE FROM THE FILE INTO A GLOBAL BYTE ARRAY
    iFileNumber = FreeFile
    Open sFileName For Binary Access Read As #iFileNumber
    Get #iFileNumber, , yDataFlashImage
    Close #iFileNumber

    '// FOR CLARITY, WITHOUT USING CONSTANTS
    iNumberOfRows = &H6C0 \ 32  '54 Rows

    '// PUT DEVICE INTO ROM MODE
    lError = WriteSMBusInteger(&H0, &HF00)
    DoDelay 0.01

    '// ERASE DATA FLASH, ROWS ARE ERASED IN PAIRS
    For iRow = 0 To iNumberOfRows - 1 Step 2

        lError = WriteSMBusInteger(&H11, iRow)
        DoDelay 0.04

    Next iRow

    '// WRITE EACH ROW

    For iRow = 0 To iNumberOfRows - 1

        '// Set the row to program into the first element of the 33 byte array

        yRowData(0) = iRow

        '// Copy data from the full array to the row array
        For iIndex = 0 To 31
            yRowData(iIndex + 1) = yDataFlashImage((iRow * 32) = iIndex)
        Next iIndex

        '// Write the row. Length is 33 because first byte is row number
        lError = WriteSMBusByteArray(&H10, yRowData, 32 + 1)
        DoDelay 0.02

    Next iRow

    '// EXECUTE GAS GAUGE PROGRAM
    lError = WriteSMBusCommand(&H8)

End Function
```

## 5    Calibration

Devices in the bq20z70/bq20z90 family of advanced gas gauges are quick and easy to calibrate. It only takes about 5 seconds to accurately calibrate current offset, voltage, temperature, and board offset. With the Impedance Track™ devices, most calibration routines have been incorporated into firmware algorithms, which can be initiated with SMBus commands. The hardware for calibration is also simple. One current source, one voltage source, and one temperature sensor are all that is required. The accuracy of the sources is not important, only their stability. However, accurately calibrated reference measurement equipment should be used for determining the actual arguments to the function. For periodic voltage measurement, a DVM with better than 1-mV accuracy is required.

The elapsed time for calibration can be changed by modifying values in the data flash, but this is not recommended. Use the default values for the times in DF.Calibration.Config

In the CalibrateAll( ) function, command 0x51 is used to setup a current offset, voltage, current , and temperature calibration of the device.. Pack voltage calibration is generally not performed because its accuracy is not required for standard applications. In this case, Pack Voltage refers to a separate measurement of the voltage at the pack terminal and is unrelated to the SBS.Voltage( ) measurement.

The definition of the bits in command 0x51 are:

| | | | | |
|---|---|---|---|---|
| Bit 0 | Coulomb Counter Offset | | Bit 8 | Pack Gain |
| Bit 1 | Reserved | | Bit 9 | Pack Voltage |
| Bit 2 | ADC Offset | | Bit 10 | AFE Error |
| Bit 3 | Temperature, Internal | | Bit 11 | Reserved |
| Bit 4 | Temperature, External 1 | | Bit 12 | Reserved |
| Bit 5 | Temperature, External 2 | | Bit 13 | Reserved |
| Bit 6 | Current | | Bit 14 | Run ADC Task Continuously |
| Bit 7 | Voltage | | Bit 15 | Run CC Task Continuously |

Bits 14 and 15 should always be set. These cause the Coulomb Counter and ADC tasks to run continuously, just as they do in normal operation. This has been found to increase the accuracy of the calibration.

After command 0x51 is issued, the calibration sequence is started in the firmware of the gas gauge. The calibrations are run in sequence starting from the least significant bit. Then, command 0x52 is used to poll these bits, which change from high to low as the tasks are completed. However, bits 14 and 15 do not change; hence, the masking of them in the polling loop.

It can be seen from this code that a simple modification to command 0x51 would allow it to work as a single function calibration. For example, to only calibrate voltage, only bit 7 could be set.

```
Function CalibrateAll(iVoltage As Integer, iCurrent As Integer, iTemperature As Integer, iCells
As Integer) As Long

'// iVoltage is in millivolts
'// iCurrent is in milliamps (normally negative, such as -2000)
'// iTemperature is in Kelvin/10 units, so the argument is: 10 * (Celsius + 273.15)

    Dim lError As Long
    Dim bDoingCal As Boolean
    Dim iValue As Long

    '// GO TO CALIB MODE
    IError = WriteSMBusInteger(&H0, &H40)

    '// WRITE THE NUMBER OF CELLS
    IError = WriteSMBusInteger(&H63, iCells)

    '// WRITE THE ACTUAL VOLTAGE, CURRENT & TEMPERATURE
    IError = WriteSMBusInteger(&H60, iCurrent)
    IError = WriteSMBusInteger(&H61, iVoltage)
    IError = WriteSMBusInteger(&H62, iTemperature)

    '// START CALIBRATION
    '// Useful cal lo byte  &HD5 - External temperature sensor 1
    '//                     &HF5 - External temperature sensor 1 and 2
    '//                     &HCD - Internal temperature sensor
    IError = WriteSMBusInteger(&H51, &HC0D5)

    '// POLL CALIBRATION STATUS - WAIT FOR LOWER 14 BITS TO ALL CLEAR
    bDoingCal = True
    While bDoingCal
        IError = ReadSMBusUnsignedInteger(&H52, IValue)
        bDoingCal = IValue And &H3FFF
        DoDelay 0.2 '// check every 200 millisecond
```

```
        Wend

        '// TRANSFER RESULTS TO DATAFLASH
        lError = WriteSMBusCommand(&H72)
        DoDelay 0.1 '// Insure write process is finished

        '// EXIT CALIB MODE
        lError = WriteSMBusCommand(&H73)

        End Function
```

Because of the simplified single ground system in the bq20z70/bq20z90 family, each unit should be calibrated for board offset. Use the following function in normal mode to calibrate the board offset. During this procedure, the device under test must be powered from the cells only and no external load or charge current may be applied. Note that the function requires the sense resistor value in milliohms as an argument.

```
Function CalibrateBoardOffset(fSenseMilliohms As Single) As Long
'// Device under test must be powered from the Cell side, which
'// allows the device current to flow through the sense resistor.
'// Insure no other current is flowing through the sense resistor.

    Dim lError As Long
    Dim lValue As Long
    Dim i As Integer
    Dim iExternalOffset As Integer
    Dim iInternalOffset As Integer
    Dim iBoardOffset As Long
    Dim yData(32) As Byte
    Dim iLen As Integer

    '// READ EXTERNAL OFFSET CURRENT
    lError = WriteSMBusInteger(&H40, &H8042) '//Set address of coulomb counter
    DoDelay 0.2 '// Extra settling time to clear the decimation filter

    For i = 1 To 4
        DoDelay 0.3 '// take 4 samples at 300 ms intervals
        lError = ReadSMBusUnsignedInteger(&H42, lValue) '// Peek Coulomb Counter
        iExternalOffset = iExternalOffset + lValue
    Next i

    '// READ INTERNAL OFFSET CURRENT
    lError = WriteSMBusInteger(&H40, &H8040) '//Set address of coulomb counter config register
    lError = WriteSMBusInteger(&H41, &H43) '//Change configuration to internal mode
    lError = WriteSMBusInteger(&H40, &H8042) '//Set address of Coulomb Counter
    DoDelay 0.2 '// Extra settling time to clear the decimation filter

    For i = 1 To 4
        DoDelay 0.3
        lError = ReadSMBusUnsignedInteger(&H42, lValue) '// Read Coulomb Count
        iInternalOffset = iInternalOffset + lValue
    Nexti

    lError = WriteSMBusInteger(&H40, &H8040) '//Set address of coulomb counter config register

    lError = WriteSMBusInteger(&41, &H2) '//Return configuration to external mode

    '// CALCULATE BOARD OFFSET

    iBoardOffset = Int(16 * (iExernalOffset – iInternalOffset) + (3745 * fSenseMilliohms/1000))

    If iBoardOffset < 0 Then iBoardOffset = 65536 + iBoardOffset '// fix negative case

    '// WRITE BOARD OFFSET TO DATA FLASH. FROM DATA MANUAL, SUBCLASS=104, OFSET=16
    lError = WriteSMBusInteger(&H77, 104) '//Set subclass to 104
    lError = ReadSMBusByteArray(&H78, yData(), iLen) '// Read the page

    yData(16) = (iBoardOffset And &HFF00) \ 256 '// Modify MS byte
    yData(17) = iBoardOffset And &HFF '// Modify LS byte

    lError = WriteSMBusByteArray(&H78, yData(), iLen) '//Write page back to flash
    DoDelay 0.1 '// Insure flash write is finished

End Function
```

## 6 Writing Pack-Specific Data Flash Locations

The third step is to fine tune the data flash a little for each pack, to give it a unique identity. In the following example, the pack Serial Number is written using subclass and offset information found in the gas gauge product data sheet. Modifications to single data flash locations normally require a block read of the 32-byte data flash page, then updating the desired element of the block, and writing it back to the device. This procedure is documented in the product data sheet.

```
Function WritePackSerialNumber(iSerialNumber As Integer) As Long

    Dim lError As Long
    Dim yData(32) As Byte
    Dim iLen As Integer

    '// SET THE SUBCLASS TO 48 (FOUND IN PRODUCT DATASHEET)
    lError = WriteSMBusInteger(&H77, 48)

    '// READ THE PAGE
    lError = ReadSMBusByteArray(&H78, yData(), iLen)

    '// REPLACE THE TWO BYTES AT OFFSET 12 (FOUND IN DATASHEET) WITH NEW S/N
    yData(12) = (iSerialNumber And &HFF00) \ 256 '// modify MS byte
    yData(13) = iSerialNumber And &HFF '// modify LS byte

    '// WRITE THE PAGE BACK TO FLASH
    lError = WriteSMBusByteArray(&H78, yData(), iLen)

    '// FLASH WRITES ARE SLOW
    DoDelay 0.1

End Function

Sub DoDelay(fWaitTime As Single)

    Dim vTime As Variant

    vTime = Timer
    While Timer < (vTime + fWaitTime)

        '// fix midnight problem
        If Timer < vTime Then Exit Sub

        '// Yield to various Windows events while the delay is in progress
        DoEvents

    Wend

End Sub
```

**IMPORTANT NOTICE**

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

| **Products** | | **Applications** | |
|---|---|---|---|
| Amplifiers | amplifier.ti.com | Audio | www.ti.com/audio |
| Data Converters | dataconverter.ti.com | Automotive | www.ti.com/automotive |
| DSP | dsp.ti.com | Broadband | www.ti.com/broadband |
| Interface | interface.ti.com | Digital Control | www.ti.com/digitalcontrol |
| Logic | logic.ti.com | Military | www.ti.com/military |
| Power Mgmt | power.ti.com | Optical Networking | www.ti.com/opticalnetwork |
| Microcontrollers | microcontroller.ti.com | Security | www.ti.com/security |
| | | Telephony | www.ti.com/telephony |
| | | Video & Imaging | www.ti.com/video |
| | | Wireless | www.ti.com/wireless |

Mailing Address:     Texas Instruments

Post Office Box 655303 Dallas, Texas 75265

Copyright © 2006, Texas Instruments Incorporated